Node.js 技术分享

—刘晓阳

Node.js Everywhere

生态圈

- Node.js 使用量每年的增长达到了 100%
- 每一天新增的 npm package 达到了 400 个
- 社区活跃
- Visual Studio Code & TypeScript

使用 Node 构建桌面应用

- 跨平台
- 钉钉（NW.js） Atom（electron）

API with Node.js

- JSON-JavaScript Object Notation
- Geteway

other

- 开源硬件 Raspberry PI
- V8 GC Logs

Nodejs之多进程

槽点

- 单进程，单线程，无法利用多核CPU
- 可靠性低
- 对程序健壮性要求比较高

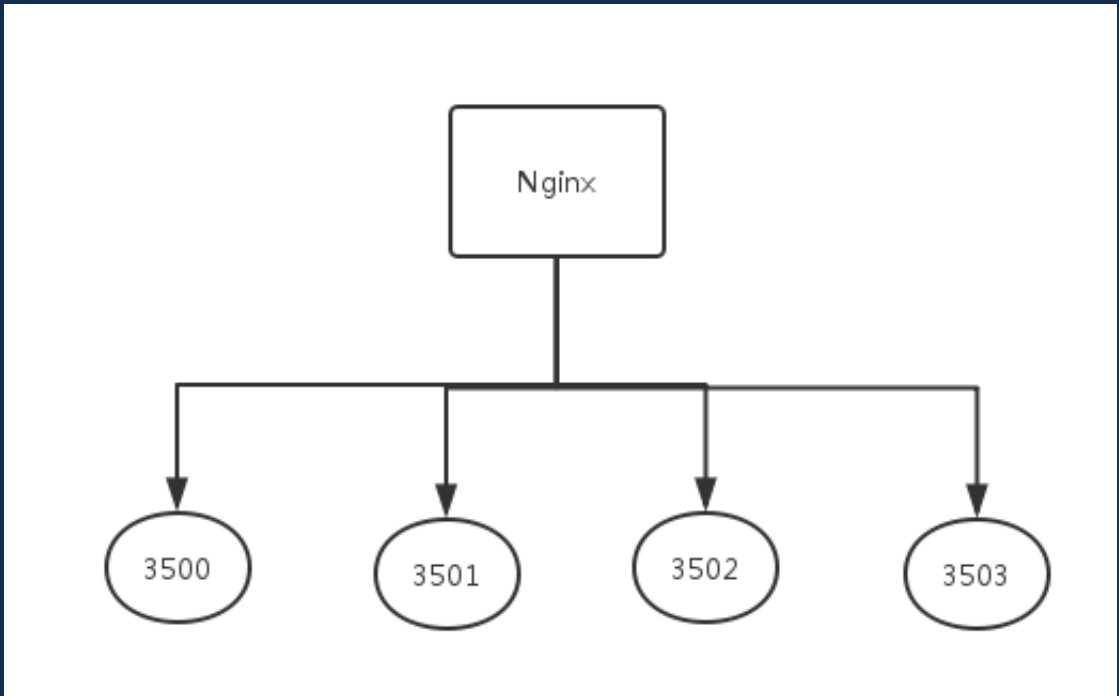
Nodejs如何实现多线程?

```
var http = require('http');

http.createServer(function(req, res){
  res.writeHead(200, {'Content-Type' : 'text/plain'});
  res.end('hello world');
}).listen(3000, '127.0.0.1');
```

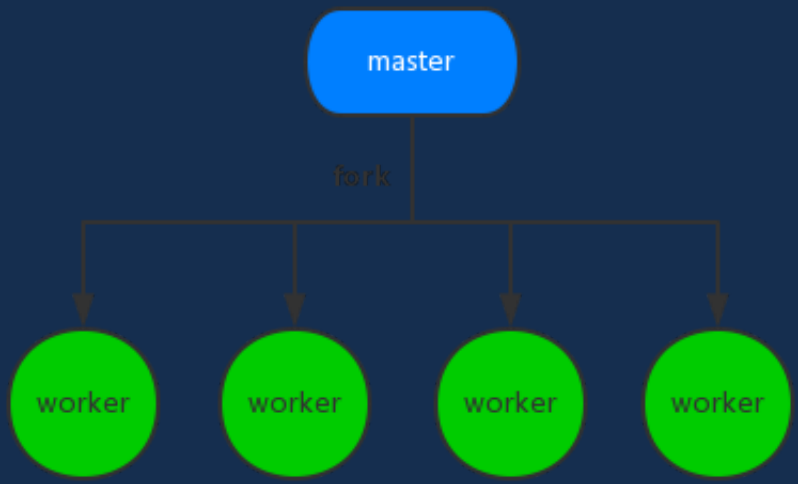
node app

```
events.js:160
      throw er; // Unhandled 'error' event
      ^

Error: listen EADDRINUSE 127.0.0.1:3000
    at Object.exports._errnoException (util.js:896:11)
    at exports._exceptionWithHostPort (util.js:919:20)
    at Server._listen2 (net.js:1246:14)
    at listen (net.js:1282:10)
    at net.js:1392:9
    at _combinedTickCallback (internal/process/next_tick.js:77:11)
    at process._tickCallback (internal/process/next_tick.js:98:9)
    at Function.Module.runMain (module.js:577:11)
    at startup (node.js:159:18)
    at node.js:444:3
```

Nginx proxy



master-worker



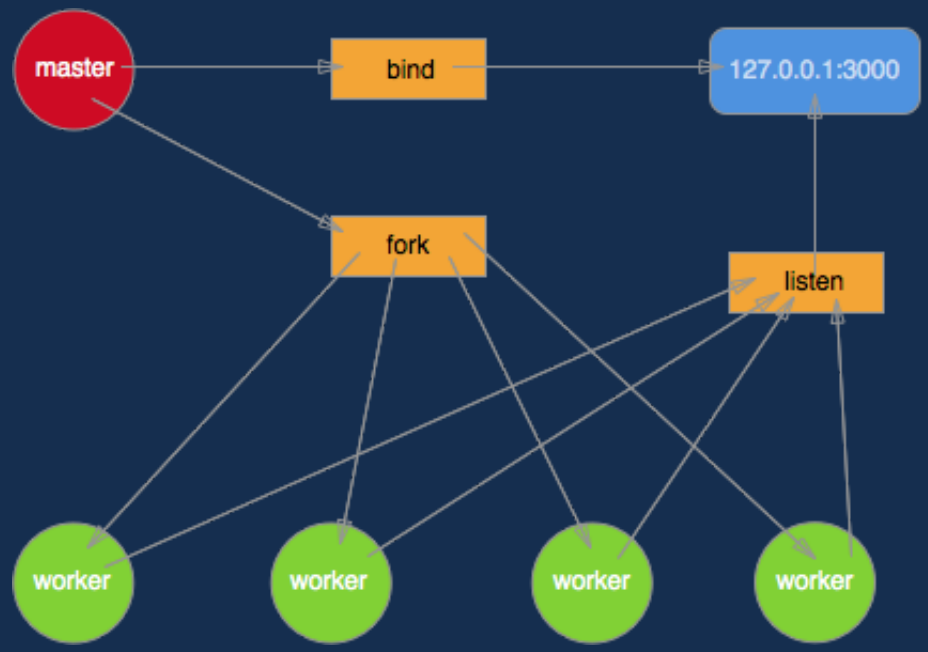
demo

```
//master.js
const net = require('net');
const fork = require('child_process').fork;

var handle= net._createServerHandle('0.0.0.0', 3000);

for(let a =0; a < 4; a++){
  fork('./worker.js').send({}, handle);
}
```

demo

```
//worker.js
const net = require('net');
process.on('message', function(m, server){
    server.listen();
    server.onconnection = function(err, handle){
    console.log('got a connection on worker, pid = %d', process.pid);
    var socket = new net.Socket({
      handle: handle
    });
    socket.readable = socket.writable = true;
    socket.end('hello js');
  }
});
```

多进程监听同一端口的进程模型



有问题吗？

- 多个进程会竞争 accpet 一个连接。
- 无法控制请求由哪个进程去处理，导致各个worker之间负载不均衡。

由master进程负责监听和调度任务

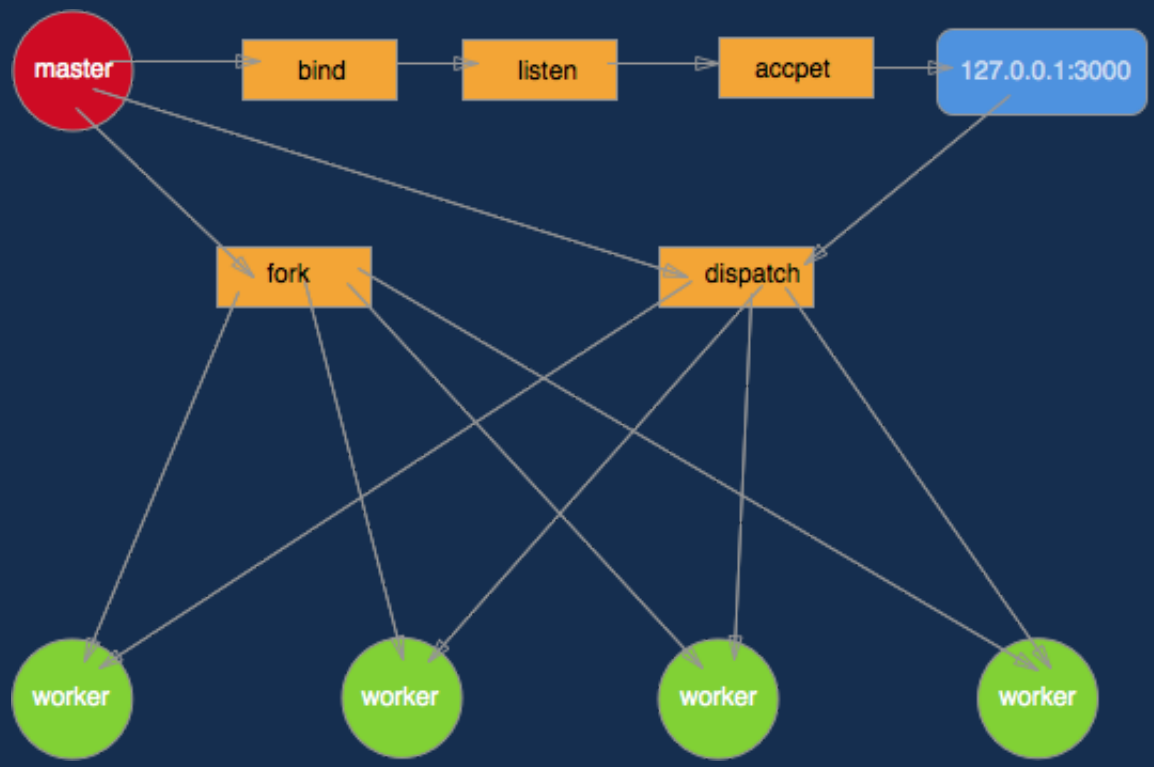
demo

```
//master.js
const net = require('net');
const fork = require('child_process').fork;

var workers = [];

for(let i = 0; i < 4; i++) {
  workers.push(fork('./worker.js'));
}

var serverHandle = net._createServerHandle('0.0.0.0', 3000);
serverHandle.listen();


serverHandle.onconnection = function(err, handle) {
  var worker = workers.pop();
  worker.send({}, handle);
  workers.unshift(worker);
}
```

demo

```
const net = require('net');
process.on('message', function(m, handle){
  console.log('got a connection on work , pid = %d', process.pid);

  var socket = new net.Socket({
    handle : handle
  });


  socket.end('hello world');
});
```

进程模型如下图


master
bind
listen
accpet
127.0.0.1:3000
fork
dispatch
worker
worker
worker
worker

进程守护

master 进程除了负责接收新的连接，分发给各 worker 进程处理之外，还得像天使一样默默地守护着这些 worker 进程，保障整个应用的稳定性。一旦某个 worker 进程异常退出就 fork 一个新的子进程顶替上去。

关于 uncaughtException

- 我们常说的程序崩了，什么是崩了
- 什么是uncaughtException
- 如何优雅的退出

线上问题

如果线上出了这样的FATAL会发生什么?

```
[2016-05-27 18:01:00.958] [FATAL] access - [Error: Can't set headers after they are sent.]
Error: Can't set headers after they are sent.
  at ServerResponse.OutgoingMessage.setHeader (http.js:689:11)
  at ServerResponse.header (/home/www/beeper_api/node_modules/express/lib/response.js:595:10)
  at ServerResponse.send (/home/www/beeper_api/node_modules/express/lib/response.js:143:12)
  at ServerResponse.json (/home/www/beeper_api/node_modules/express/lib/response.js:229:15)
  at /home/www/beeper_api/controllers/api/config.js:312:16
  at Request._callback (/home/www/beeper_api/lib/request_api.js:55:10)
  at Request.self.callback (/home/www/beeper_api/node_modules/request/request.js:123:22)
  at Request.EventEmitter.emit (events.js:98:17)
  at Request.<anonymous> (/home/www/beeper_api/node_modules/request/request.js:1047:14)
  at Request.EventEmitter.emit (events.js:117:20)
  at IncomingMessage.<anonymous> (/home/www/beeper_api/node_modules/request/request.js:998:12)
  at IncomingMessage.EventEmitter.emit (events.js:117:20)
  at _stream_readable.js:919:16
  at process._tickDomainCallback (node.js:463:13)
```

demo

```
function demo (req, res) {
    res.json({code : 1, msg : 'err'});
    return res.json({code : 1, msg : 'err2'});
}
```

demo

```
function demo (req, res) {

    request_api_lib.request_get(url, function(){
        if(err) {
            res.json({code : 1, msg : 'err'});
        }


        return res.json({code : 1, msg : 'err2'});
    });
}
```

后果很严重

```
[log_user@yn-log-server211 beeper_api]$ grep 'exit process on timeout' --count  beeper_api_access_20160421.log
5912
[log_user@yn-log-server211 beeper_api]$
[log_user@yn-log-server211 beeper_api]$
```

- 6000 * 3 = 18000

Seq

```
function create_pack_report(query, callback) {
    var customer_id = +_.get(query, 'customer_id', '');

    new Seq()
        .par('charge', function () {
            _get_pack_charge_from_mt(customer_id, this);
        })
        .par('bu_leader', function (){
            admin_user_lib.get_bu_leader_by_cuid(customer_id, this);
        })
        .par('customer', function () {
            customer_lib.get_customer_info(customer_id, this);
        })
        .par('pack_config', function(){
            _get_pack_config_from_mt(this);
        })
        .seq(function () {
            var charge = this.vars.charge || {},
                bu_leader = this.vars.bu_leader || {},
```

- 有问题吗?

undefined.name

- customer && customer.name || ''
- _.get(customer, 'name', '')

当我们比较语言优劣，容易局限在语言本身，而忽视了配套的一些关键因素

Thank you for guys